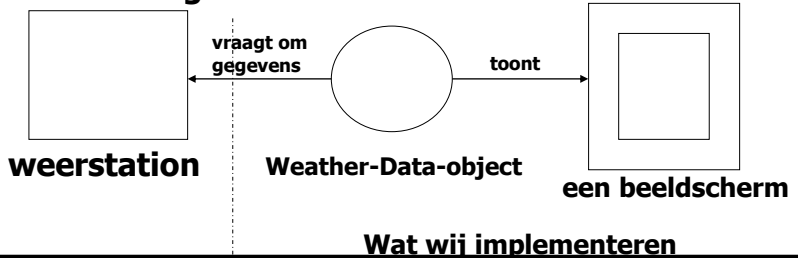
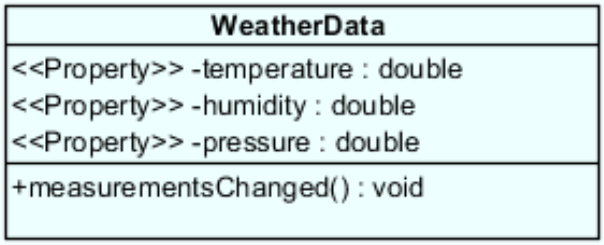
Observer – gedrag van objecten

Voorbeeld "Applicatie weerstation"

- **weerstation (het fysieke apparaat dat de actuele weergegevens ophaalt),**
- **het Weather-Data-object (dat de gegevens verwerkt die van het weerstation komen)**
- **drie schermen (toont actuele weersgesteldheid, weerstatistieken en weersverwachting)**
- **? Toekomstige schermen**



Oplossing?:



- De klasse heeft get-methoden voor de drie gemeten waarden: temperatuur, vochtigheid en luchtdruk.
- De methode measurementsChanged() wordt iedere keer aangeroepen wanneer er nieuwe weermetingen beschikbaar zijn.
- We dienen ook drie schermelementen te implementeren.
- Het systeem moet kunnen worden uitgebreid.

Oplossing?:

OOAD

```
public class WeatherData
{
    private double temperature, humidity, pressure;
    ...

    public double getHumidity() { return humidity;}
    public double getPressure() { return pressure;}
    public double getTemperature() {return temperature;}

    public void measurementsChanged()
    {
        double temp = getTemperature();
        double humidity = getHumidity();
        double pressure = getPressure();

        currentConditionsDisplay.update(temp, humidity, pressure);
        statisticsDisplay.update(temp, humidity, pressure);
        forecastDisplay.update(temp, humidity, pressure);
    }
        ...
```

Verzamel de recenste metingen

update nu de schermen

Wat is er verkeerd aan deze implementatie? Denk nog eens terug aan de concepten en principes uit "inleiding tot design patterns".

```
public void measurementsChanged()
{
    double temp = getTemperature();
    double humidity = getHumidity();
    double pressure = getPressure();

    currentConditionsDisplay.update(temp, humidity, pressure);
    statisticsDisplay.update(temp, humidity, pressure);
    forecastDisplay.update(temp, humidity, pressure);
}
```

Door naar concrete implementaties te coderen beschikken we over geen enkele manier om andere schermelementen toe te voegen of te verwijderen zonder wijzigingen in het programma aan te brengen.

Ze hebben allemaal een methode update() die de waarden voor temp, humidity en pressure ophaalt. We zullen een gemeenschappelijke interface kunnen gebruiken.

Maak kennis met het Observer Pattern

vb. kranten en tijdschriftenabonnementen

1) Een krantenuitgeverij begint met het uitgeven van kranten

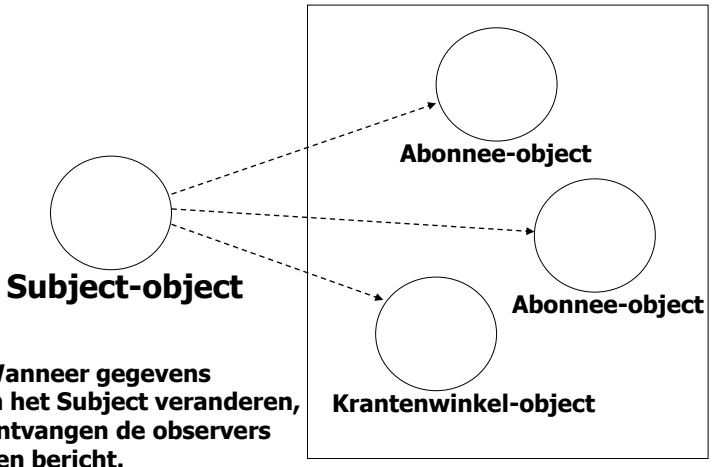
2) Je abonneert je bij een bepaalde uitgever en iedere keer als er een nieuwe editie verschijnt, wordt die bij je bezorgd. Zolang je abonnee blijft, ontvang je kranten.

3) Als je geen kranten meer wil, zeg je je abonnement op en zij stoppen met het bezorgen.

4) Zolang de uitgever actief blijft, abonneren mensen, hotels, luchtvaartmaatschappijen en andere bedrijven zich voortdurend op de krant of zeggen hun abonnement op.

Uitgevers + Abonnees = Observer Pattern
uitgever = SUBJECT
abonnee = OBSERVER

OOAD



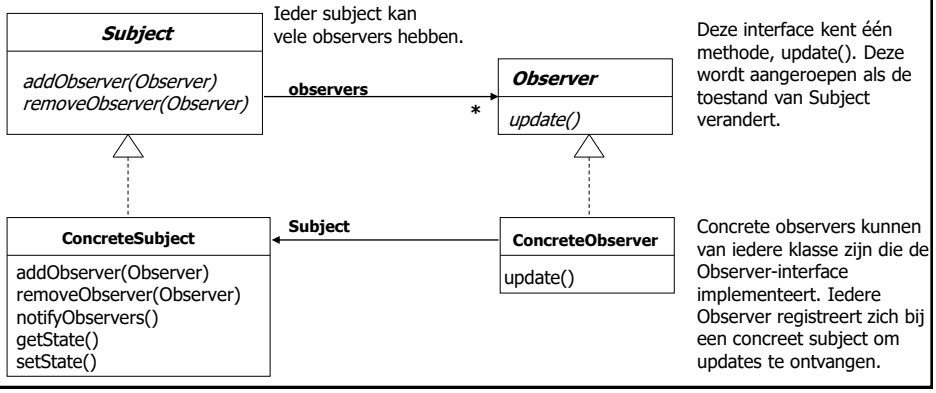
De observers hebben een abonnement (zijn geregistreerd) bij het Subject om updates te ontvangen wanneer de gegevens van het Subject veranderen.

Wanneer gegevens in het Subject veranderen, ontvangen de observers een bericht.

Observer – gedrag van objecten

OOAD

Het Observer Pattern definieert een één-op-veel-relatie tussen objecten, zodanig dat wanneer de toestand van een object verandert, alle afhankelijke objecten worden bericht en automatisch worden geüpdatet.



Ieder subject kan vele observers hebben.

Deze interface kent één methode, update(). Deze wordt aangeroepen als de toestand van Subject verandert.

Concrete observers kunnen van iedere klasse zijn die de Observer-interface implementeert. Iedere Observer registreert zich bij een concreet subject om updates te ontvangen.

De kracht van zwakke koppeling

OOAD

- Wanneer twee objecten een zwakke koppeling hebben, dan kunnen ze interactie plegen, maar weten ze weinig over elkaar.

- Het Observer Pattern zorgt voor een ontwerp van objecten waarin subjecten en observers zwak gekoppeld zijn.
  - Het enige wat een subject over een observer weet is dat deze een bepaalde interface implementeert (de Observer-interface).
  - We kunnen op ieder moment nieuwe observers toevoegen.
  - We hoeven het subject nooit te veranderen om nieuwe soorten observers toe te voegen.
  - We kunnen subjecten en observers onafhankelijk van elkaar hergebruiken.
  - Veranderingen in het subject of een observer hebben geen invloed op elkaar.

OOAD

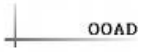
De kracht van zwakke koppeling

Streef naar ontwerpen met een zwakke koppeling tussen de objecten die samenwerken.

Zwak gekoppelde ontwerpen stellen ons in staat flexibele OO-systemen te bouwen die met verandering om kunnen gaan, omdat ze de wederzijdse afhankelijkheid tussen objecten erg klein maken.

10

Terug naar het project Weerstation

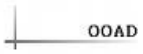


We gaan het Observer Pattern gebruiken. Maar hoe gaan we dit toepassen?

> Het Observer Pattern definieert een één-op-veel-relatie tussen objecten, zodanig dat wanneer de **toestand** van een object verandert, alle afhankelijke objecten worden bericht en automatisch worden geüpdatet.

"**toestand**"?
Hoe krijgen we de weermetingen in de schermelementen?
Ieder schermelement kan anders zijn.

Terug naar het project Weerstation



"**toestand**"?
De klasse WeatherData heeft zeker een toestand… dat is de temperatuur, de luchtvochtigheid en de luchtdruk – deze veranderen zeker.

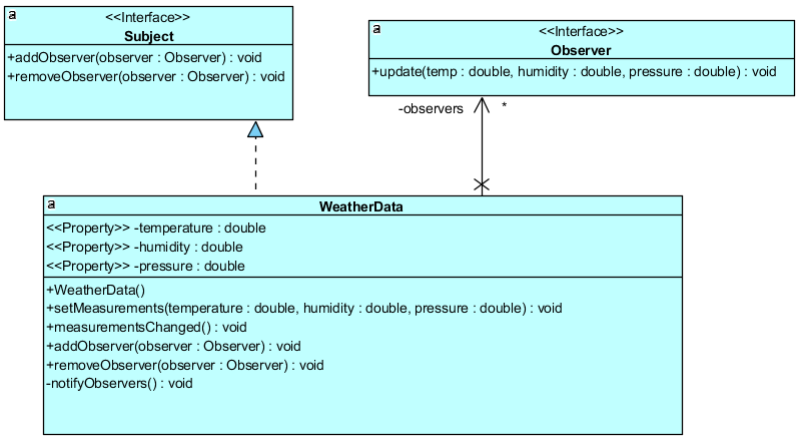
Hoe krijgen we de weermetingen in de schermelementen?
Van object WeatherData maken we een **subject** en de schermelementen **observers**. De schermen registreren zich bij het object WeatherData om de informatie te krijgen die ze willen hebben.

Ieder schermelement kan anders zijn.
Een gemeenschappelijke interface gebruiken. Ook al heeft iedere component een ander type, ze moeten allemaal dezelfde interface implementeren, zodat het object Weather-Data weet hoe het de meetwaarden naar ze moet versturen.

Terug naar het project Weerstation

OOAD

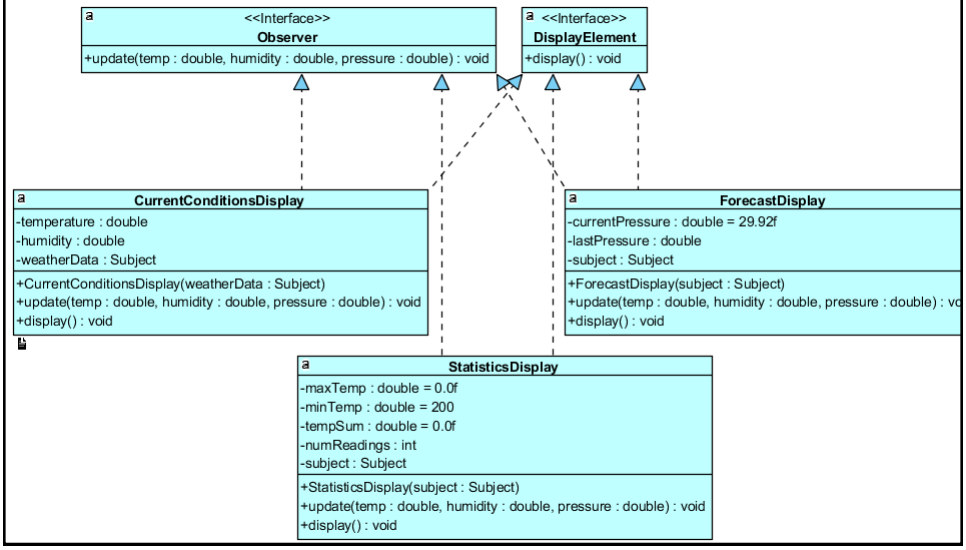
package domein:



Terug naar het project Weerstation

OOAD

package gui:



Implementatie van het weerstation

OOAD

```
public interface Subject
{
        public void addObserver(Observer observer);
        public void removeObserver(Observer observer);
}


public interface Observer
{
        public void update(double temp, double humidity, double pressure);
}


public interface DisplayElement
{
        public void display();
}
```

Dit zijn de toestandswaarden die de Observer van het Subject krijgt, wanneer een meetwaarde verandert.

Implementatie van de Subject-interface in WeatherData

OOAD

```
…
public class WeatherData implements Subject
{
        private double temperature, humidity, pressure;
        private Set<Observer> observers;

        public WeatherData()
        {
                observers = new HashSet<>();
        }

        @Override
        public void addObserver(Observer observer) {
                observers.add(observer);
        }

        @Override
        public void removeObserver(Observer observer) {
                observers.remove(observer);
        }
```

We hebben een Set toegevoegd om de Observers in op te slaan, en we creëren deze in de constructor.

```
public void setMeasurements(double temperature, double humidity,
                            double pressure) {
        this.temperature = temperature;
        this.humidity = humidity;
        this.pressure = pressure;
        notifyObservers();
}

public double getHumidity() {
        return humidity;
}

public double getPressure() {
        return pressure;
}

public double getTemperature() {
        return temperature;
}
```

We berichten de Observers wanneer we bijgewerkte meetwaarden van het weerstation hebben ontvangen.

Implementatie van de Subject-interface in WeatherData

```
private void notifyObservers() {
        observers.forEach(observer ->
                observer.update(temperature, humidity, pressure));

}
}
```

Hier informeren we alle observers over de toestand . Omdat het allemaal Observers zijn, weten we dat ze allemaal update() hebben geïmplementeerd, dus weten we hoe we ze een bericht kunnen sturen.

We gaan nu de schermelementen maken


```
public class CurrentConditionsDisplay implements Observer, DisplayElement
{
        private Subject weatherData;
        private double temperature, humidity;

        public CurrentConditionsDisplay(Subject weatherData)  {
                this.weatherData = weatherData;
                weatherData.addObserver(this);
        }

        public void update(double temp, double humidity, double pressure) {
                this.temperature = temp;
                this.humidity = humidity;
                display();
        }
        public void display() {
                System.out.printf("Actuele weergesteldheid %.1f graden en
%.1f %% luchtvochtigheid\n",   temperature, humidity);
        }
}
```

Het observerobject wordt in zijn constructor overgedragen aan het object WeatherData(het Subject). We gebruiken deze om het scherm als een observer te registreren.

Het weerstation in gebruik nemen

```
public class StartUp
{
        public static void main(String args[])
        {
                new WeatherStation();
        }
}
public class WeatherStation
{
        public  WeatherStation()
        {
                System.out.println("Weerstation");
                WeatherData weatherData = new WeatherData();

                CurrentConditionsDisplay currentDisplay =
                        new CurrentConditionsDisplay(weatherData);
                StatisticsDisplay statisticsDisplay = new
                        StatisticsDisplay(weatherData);
                ForecastDisplay forecastDisplay = new ForecastDisplay(weatherData);

                weatherData.setMeasurements(80, 65, 30.4f);
                weatherData.setMeasurements(82, 70, 29.2f);
                weatherData.setMeasurements(78, 90, 29.2f);
        }
}
```

```
Weerstation
Actuele weergesteldheid 80.0 graden en 65.0 % luchtvochtigheid
Gem/Max/Min temperature = 80.0/80.0/80.0
Weersverwachting: Beter weer op komst!
Actuele weergesteldheid 82.0 graden en 70.0 % luchtvochtigheid
Gem/Max/Min temperature = 81.0/82.0/80.0
Weersverwachting: Koeler, regenachtig weer op komst
Actuele weergesteldheid 78.0 graden en 90.0 % luchtvochtigheid
Gem/Max/Min temperature = 80.0/82.0/78.0
Weersverwachting: Meer van hetzelfde
```

Hoe het ingebouwde Observer Pattern van Java gebruiken

Zie cursus JAVA.

De java.util implementatie van Observer/Observable is niet de enige plaats in de JDK, waar je het Observer Pattern kunt aantreffen.
JavaFx kent ook de implementatie van dit pattern.

Bv.

De knop is Subject

Observer

```
Button btn = new Button();
btn.setOnAction(new EventHandler<ActionEvent>() {
    @Override public void handle(ActionEvent event) {
        System.out.println("Hello World!"); } });
```

In plaats van update() wordt de methode handle() aangeroepen als de toestand van het subject (in dit geval de knop) verandert.

21